## Experiment – 7

**Student Name: Amogh Raina** **UID: 23BCS80098**

**Branch: BE-CSE** **Section/Group: 22BCS_IOT-639/A**

**Semester: 6th** **Date of Performance: 17/03/25**

**Subject Name: Advanced programming Lab II** **Subject Code: 22CSP-351**

**PROBLEM 1:**

1. **Aim:** Climbing Stairs **(Easy)**

2. **Objective:** You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

3. **Code:**

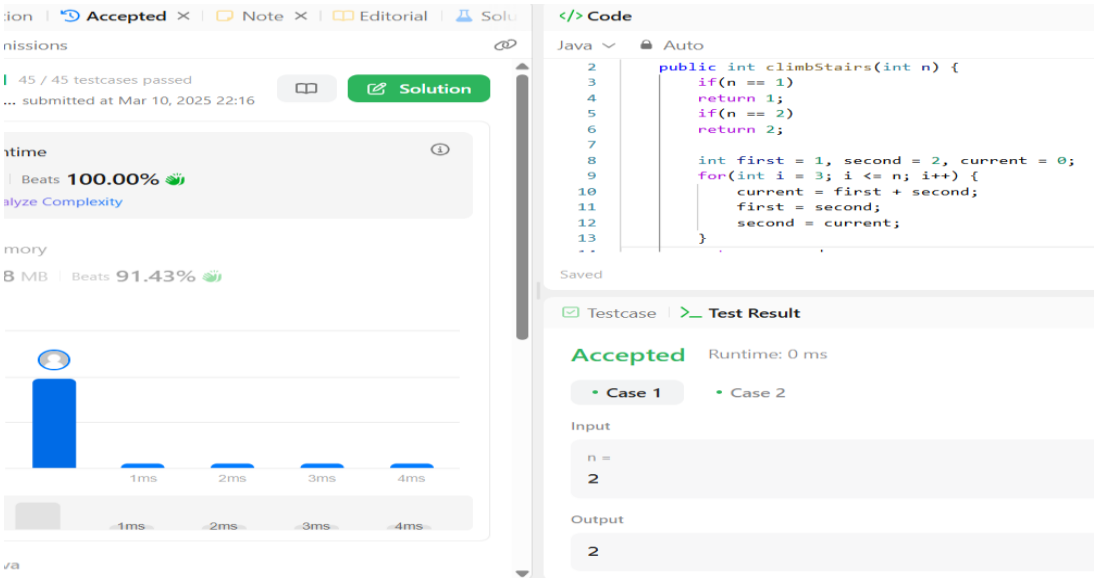
```
class Solution {
    public int climbStairs(int n) {
        if(n == 1)
        return 1;
        if(n == 2)
        return 2;

        int first = 1, second = 2, current = 0;
        for(int i = 3; i <= n; i++) {
            current = first + second;
            first = second;
            second = current;
        }
        return second;
    }
}
```

4. **Time Complexity:**

   Time complexity = O (n)

   Space complexity = O (1)

5. **Output:**




**PROBLEM 2:**

1. **Aim:** Unique Paths **(Medium)**.

2. **Objective:** There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.
   Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

3. **Code:**

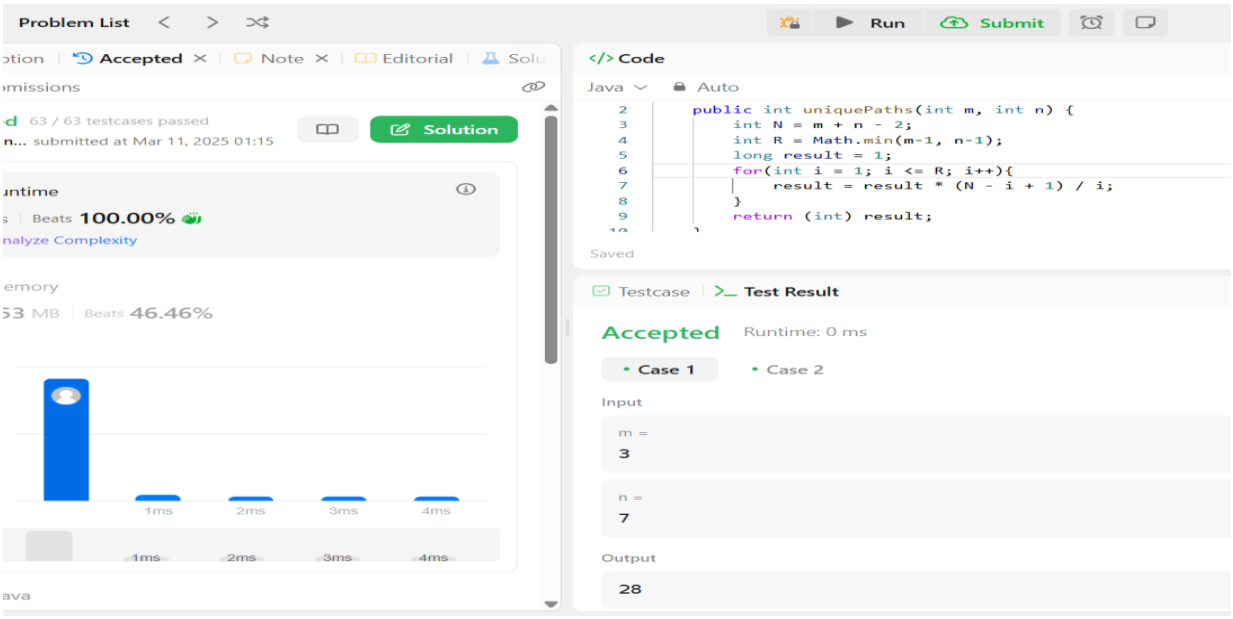
```
class Solution {
    public int uniquePaths(int m, int n) {
        int N = m + n - 2;
        int R = Math.min(m-1, n-1);
        long result = 1;
        for(int i = 1; i <= R; i++){
            result = result * (N - i + 1) / i;
        }
        return (int) result;
    }
}
```

4. **Time Complexity:**

   Time Complexity: O (min(m,n))

   Space Complexity: O (1)

5. **Output:**




**PROBLEM 3:**

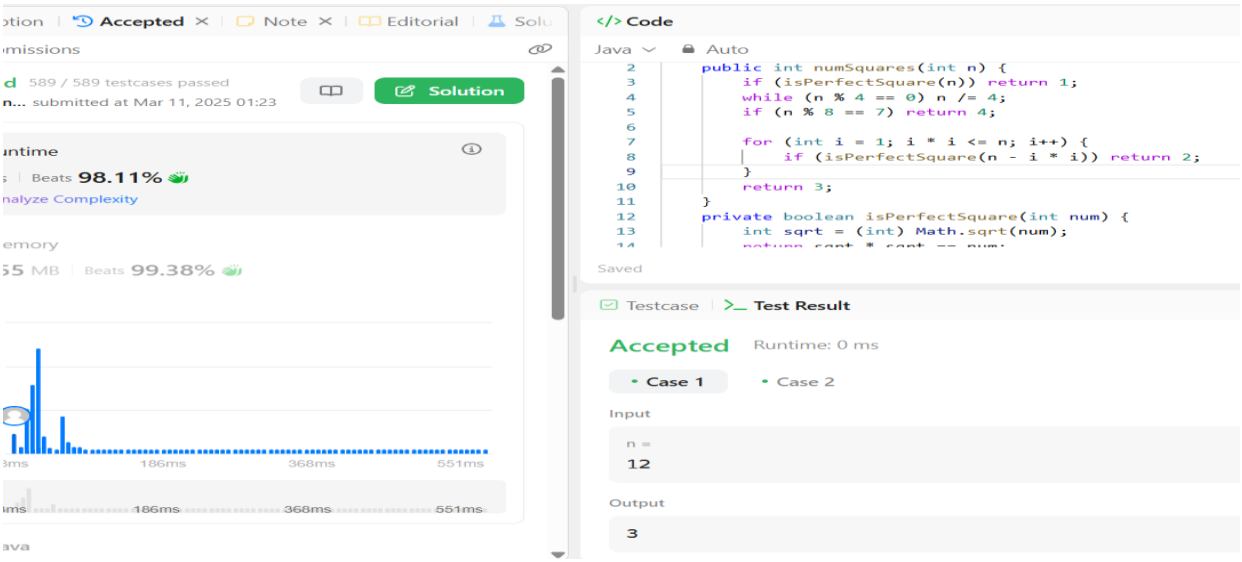
1. **Aim:** Perfect Squares **(Hard)**.

2. **Objective:** Given an integer n, return the least number of perfect square numbers that sum to n.

3. **Code:**

```
class Solution {
    public int numSquares(int n) {
        if (isPerfectSquare(n)) return 1;
        while (n % 4 == 0) n /= 4;
        if (n % 8 == 7) return 4;

        for (int i = 1; i * i <= n; i++) {
            if (isPerfectSquare(n - i * i)) return 2;
        }
        return 3;
    }
    private boolean isPerfectSquare(int num) {
        int sqrt = (int) Math.sqrt(num);
        return sqrt * sqrt == num;
    }
}
```

4. **Output:**




5. **Time Complexity:**
   Time Complexity = O (sqrt(n))  Space Complexity = O (1)

**PROBLEM 4:**

1. **Aim:** Word Break II **(Hard)**.

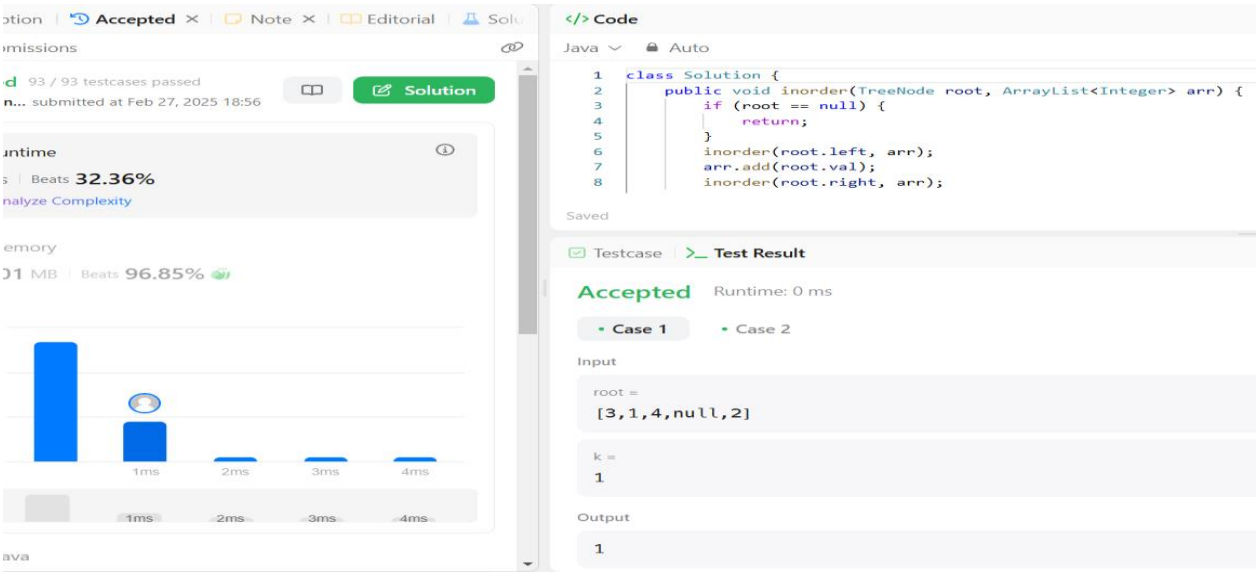
2. **Objective:** Given a string s and a dictionary of strings wordDict, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

3. **Code:**

```
import java.util.*;
class Solution {
    public List<String> wordBreak(String s, List<String> wordDict) {
        Set<String> wordSet = new HashSet<>(wordDict);
        Map<String, List<String>> memo = new HashMap<>();
        return dfs(s, wordSet, memo);
    }
    private List<String> dfs(String s, Set<String> wordSet, Map<String, List<String>> memo) {
        if (memo.containsKey(s)) return memo.get(s);
        List<String> result = new ArrayList<>();
        if (s.isEmpty()) {
            result.add("");
            return result;
        }
        for (int i = 1; i <= s.length(); i++) {
```

```
            String prefix = s.substring(0, i);
            if (wordSet.contains(prefix)) {
                List<String> suffixWays = dfs(s.substring(i), wordSet, memo);
                for (String suffix : suffixWays) {
                    result.add(prefix + (suffix.isEmpty() ? "" : " ") + suffix);
                }
            }
        }
        memo.put(s, result);
        return result;
    }
}
```

**4. Output:**




**5. Time Complexity:**

Time Complexity = O (2^n)  Space Complexity = O (n + W)

**6. Learning Outcome:**

a) Problems Covered: Climbing Stairs, Unique Paths, Perfect Squares, Word Break II.
b) Complexities: O(n), O(min(m,n)), O(sqrt(n)), O(2^n) with space complexities O(1) or O(n + W).
c) Techniques Used: Dynamic Programming, Combinatorics, Mathematical Optimization, Backtracking with Memoization.
d) Applications: Pathfinding, Staircase Counting, Number Decomposition, Sentence Segmentation.